International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A New View on Method-Calls and Contracts to Facilitate Developers in making their Design Decisions

Dr. G. Manoj Someswar[1], M. Ratna Raju[2]

[1]Professor & Research Supervisor, Mewar University, Chittorgarh, Rajasthan, India

[2]Research Scholar, Mewar University, Chittorgarh, Rajasthan, India

Abstract: ***Existing concepts are time tested and are used for the very purpose of integrity, reliability and accuracy. A comprehensive overview of these concepts helps the software developers to take appropriate decisions with a view to procure accurate results with successful implementation. Even reinventing things often opens a new angle and helps to gain a deeper understanding of the related topics. Calling a method of a component is a very basic process, but it is not as simple as it seems. Each method must be called by using the correct parameters and will often return a value or an object reference. There are a lot of possible reasons why a call may cause an error. Some of these reasons, like an incorrect signature, can be checked by the compiler. Other reasons cannot be checked by the compiler because it is not possible to specify them in the chosen language. As an instance for the problem in consideration, it is clearly indicative from the developers point of view that there is the possibility of occurrence of dependencies in the used parameters due to the reason that there will be two integer parameters and the first parameter will be necessarily greater than the second one. There are other specific reasons for the occurrence of a possible error during a call which could necessarily be a busy device, an incorrect initialized object, a reference to a deleted object etc. The best thing a developer can be able to do is to clearly specify the behavior of the object when the call cannot be finished successfully. The behavior is defined by an exception, so that it is possible to react to this error. This research paper tries to open a new view on method-calls and contracts to facilitate developers in making their design decisions in the area of software engineering and design.***

Keywords: deterministic call, nondeterministic call, method call, component, state, state machine, pseudo linear contract

1. Introduction

As a matter of fact, as against software life cycle models from the point of view of the software developers, the models pertaining to software process explicitly indicate a sequence of activities, objects, transformations, and events over a network that includes suitable relevant strategies for establishing software solutions. Keeping in view of this fact, the design and development of suitable models can be utilized to develop more accurate and generalized descriptions of software development life cycle activities. The strength and reliability emerges from their usage of a much required rich notation, syntax, or semantics which are required for computational processing and to facilitate developers in making their design decisions using method calls and contracts.

Software process networks are to be recognized as showing a comprehensive multiple interconnected and well organized task chains. These task chains indicate a notional sequence of actions which are non-linear that facilitate structural transformation and provide for transformation of available computational objects mainly the resources into intermediate or finished products. The concept of Non-linearity and its subject matter in to emulate that the sequence of activities may be non-deterministic, iterative, accommodate multiple/parallel alternatives, and also partially ordered to accommodate for incremental progress. From the point of view of a non-linear sequences of primitive actions which denote atomic units of computing work including task actions, for instance, in case of a user's selection of a command or menu entry with the help of a mouse or keyboard. This concept of task chains and formation of such associated behavior has been referred by Winograd and others to these units of cooperative work between people and computers as "structured discourses of work" while task chains have become popularized under the name of "workflow".

Task chains have to be understood thoroughly by the designers and developers so that it enables them to identify the characteristic features of prescriptive or descriptive action sequences.

Idealized plans for the implementation of prescriptive task chains are based upon the presumption that what type of actions should be accomplished and in what sequential order of priority. As an indication, a task chain for the activity of object-oriented software design might include the following task actions:

1. Development an informal narrative specification of the system.
2. Identification of the objects and their attributes.
3. Identification of the operations on the objects.
4. Identification of the interfaces between objects, attributes or operations.
5. Implementation of the operations.

However to be more precise, this process which includes a sequence of actions that are designed to provide for multiple iterations and primitive action invocations which are purely non-procedural leading in the pathway of incremental progress towards an object-oriented software design.

For the purpose of forming a complete production network or web based configuration, it is necessary that task chains join or split into other task chains. The outcome of a well-defined production web is indicative of the "organizational production system" that changes to a raw computational, cognitive and also some organizational resources into assembled, integrated and usable software systems. The very basis for the development, utilization and maintenance of a new software system is the production lattice which actually lays a strong foundation for the development of such a system. Analytically speaking, the prescriptive task chains and actions cannot be formally acknowledged to anticipate all possible circumstances or idiosyncratic foul-ups which can crop up in the real world of software development. Hence, in the case of any software production web there is every possibility that in whatever manner as such will provide for any way to realize only an approximate or incomplete description of software development.

In the case of any breakdown or inadequacy, articulation work provides for a kind of unanticipated task that is clearly indicative in such circumstances. It is work that represents an open-ended non-deterministic sequence of actions taken to restore progress on the disarticulated task chain, or else to shift the flow of productive work onto some other task chain. Thus, descriptive task chains are employed to characterize the observed course of events and situations that emerge when people try to follow a planned task sequence.

Software process dynamism is often referred to as the notion of articulation work and articulation work often is an indicative of software evolution which encompasses different actions people who utilize and accommodate to the contingent or anomalous behavior of a software system or negotiation with others who may be able to affect a system modification or otherwise bring about a notable change in the existing circumstances.

2. Problem Definition & Objectives

Software Engineering gives many opportunities to researchers to identify problem areas which are critical to the successful implementation of the projects on the client sites. The scenario described above in which multiple participants hold multiple views on the software system they are developing, may be termed “multi-perspective software development”. Software engineering methods designed to support such multi-perspective development must be able to handle the multiplicity of participants, views, development strategies and notations, in such a way that the existing method calls and contracts can be implements successfully by making suitable changes in the process models and derive probabilistic results.

The research study is being carried out keeping in view of the above problem area and developing a possible prototype which fulfills it in the best possible way by making suitable changes in the existing system models after evolving a suitable technical strategy and successfully implementing the components in real life scenarios. An elaborate study of the existing system paved way for the design and development of the proposed system in this research paper.

3. Existing System

According to the modes of operability in general there are two types of method-calls. As a matter of fact, calls that will determine successfully in both case and method-calls that can be interrupted because something is going wrong. To make it easier to distinguish between these types, they will be named deterministic call and nondeterministic call.

From the point of view of research, it is observed that in most of the occasions, developers often concentrate on the deterministic calls and forget to define the behavior of the nondeterministic ones. Both method types are equally important for the design process and also for testing the software which is the basic requirement.

Also, it is necessary to understand that it is important to accept an exception as a valid result of a call. It is to be noted that the difference between an exception and a result of a successful call is that the execution of the program will follow a different way. An invalid call would be if an error is not reported by an exception. This often causes a crash of the whole application and this fact is time tested. It is therefore imperative to give utmost importance to the above implication keeping in view of the associated risk factors [1]

From the research point of view, one basic assumption is that each component has internal states. It has been observed on several occasions that developers often do not recognize these states and hence there is no representation of these states in the program. Each component has at least two states, just Created and deleted. Depending on the functionality there may be only these two states or a lot of other states between them. Sometimes flags or status variables are used to store state information [2] Even when they are implemented, they cannot be used for testing, as there can be a mistake in the code that is used to set the state information and everything seems to be right although an error has occurred. That is why the only reliable way of testing is to monitor the behaviour of the class.

Rationally speaking, the our research work has shown that before describing the behaviour of a component by defining a sequence of calls we should take a closer look at method-calls and define a model for them which is prerequisite requirement for the successful outcome.

Considering the fact that in the case of a method namely calling a method will bring about a radical change in the internal state of a component which can result in a new state or the component that can stay in the same one. During the call, the component changes into an intermediate state in which the component calculates the result of the method call. However, it is to be made clear from the research point of view that sometimes becomes necessary to consider this state but often the component stays in this state for a very short time and therefore it can be ignored [3] Also, when a method-call is blocking, this state must be taken into consideration. By taking a further example into account for considering this intermediate state is particularly when the method is solving a concurrency problem. In this case the method has to be defined asynchronized which means that only one call of this method can be done as long as the

execution of the method has not been finished.

From the viewpoint of developers and designers, it is clearly ascertained that during the execution of a method an error can obviously occur. Such a condition will definitely make the component to switch into a state that cannot be directly reached from the original state. Our research work has shown that there seems to be a needlessly complicated view of method calls, but this model allows a simpler test-framework and it is important to understand this view of method-calls to work with the test-framework. The test-framework becomes simpler because this model defines deterministic state-machines for this nondeterministic problem of calling methods and gives a broad overview of the outcome of our research work [4].

Here are some rules for understanding the following graphics and the more enhanced ones:

- Circle: Equals a state.
- Line with arrow: Equals a transition between two states. The arrow shows the direction and points at the result state
- Solid line: Equals a normal transition. This transition is the reason why the method was written. [5]
- Dotted items: Denote internal intermediate states or transitions that are used by the model. Transitions caused by events like exceptions must be specified for error handling in order to make it possible to test the correct behavior. [6]
- Rectangle: Equals an event, like a method-call or the occurrence of an exception. It is also seen that an event can have more than one eject, however, up to now the only eject is that the transition is executed. [7]

3.1 Deterministic call

In this case, we have considered this model example for the purpose of studying the existing system to consider the fact that the component is in the state A before a method is called. During the call the component is in the intermediate state A*. After the result has been calculated there is only one possible state in this example labeled as B. This is a diagrammatic representation to assess the state of the call [8]

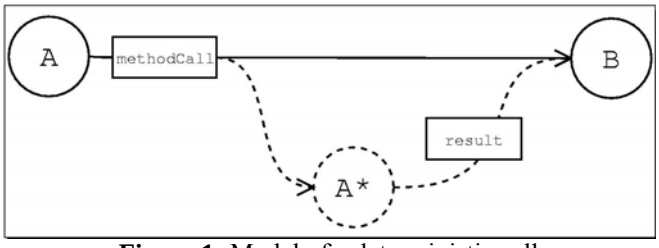

Figure 1: Model of a deterministic call

3.2 Nondeterministic call

It is found that a nondeterministic call is quite equal to a deterministic call. However the difference is that the intermediate state A* has two possible transitions into further states. Keeping this in view, once the result is calculated, and then the component will switch into state B. The component will switch into the state C only if there is a possibility of occurrence of an error. As a matter of fact, a diagrammatic representation is enough in order to assess the possible existence of states in the case of a nondeterministic call [9]

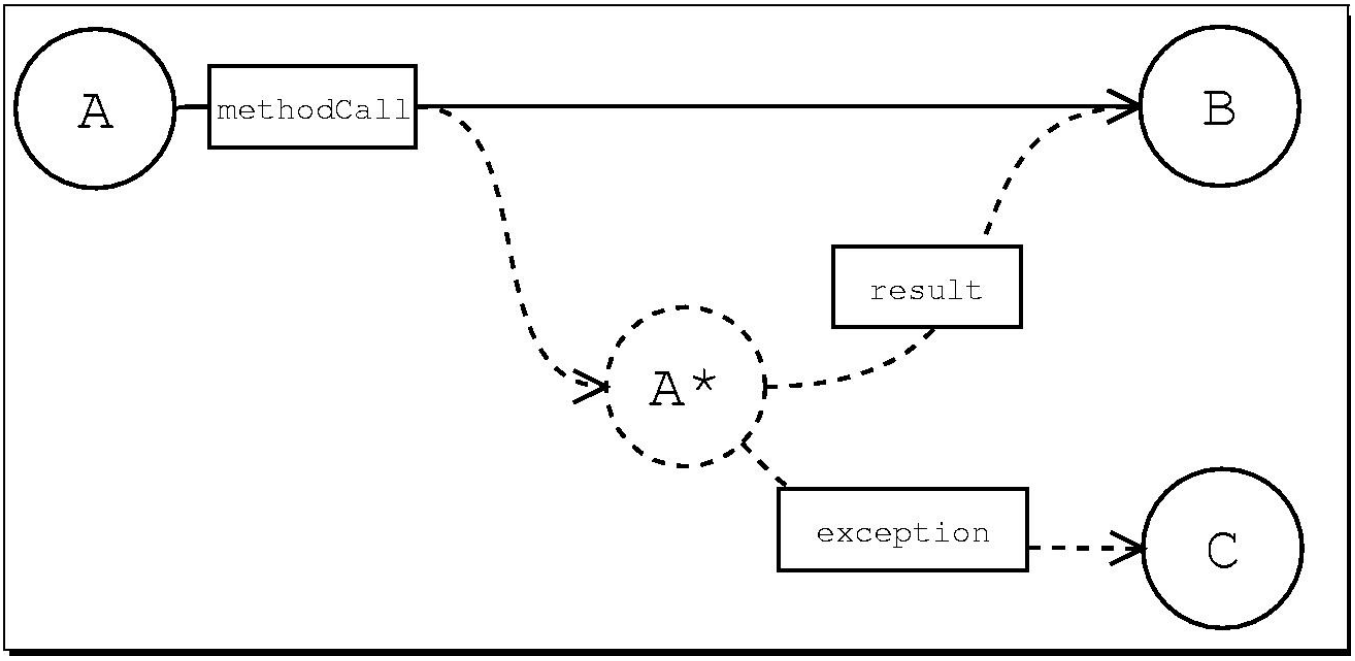

Figure 2: Model of a nondeterministic call

3.3 Sequence of Calls

For the propose of our research study, we have come to the conclusion that it is very important to describe the usage and the behavior of components both in order to use and to test the component. Therefore by thinking of the description of a component and considering the testability during the design process makes it easier to design simpler and therefore better components. If it is easy to test a component, it is probably easy to use it. Otherwise components that do not facilitate these tests often include design problems.

The behavior and the allowed usage can be described by a sequence of method-calls and their results. Some methods

cannot be called when a certain state has not been reached. These calls define a possible sequence of method-calls which is the only way the component can be used. Sequences of method calls are the first step to define contracts of components.

3.4 Contracts

As part of the design and implementation of method calls and contracts from a realistic perspective, it is ascertained that as and when a component is implemented, there is no direct way to specify its usage and behaviour. The usage is written down in the documentation of a component. It is difficult to write a program that extracts test-cases for the component from the documentation. It is to be noted that documents are human readable otherwise. A better means of giving solution to this problem is to define contracts between components. These contracts can be written in a machine readable form and therefore it s possible to transform them into human readable texts or diagrams [10]

On the other hand, one part of a contract is the way of calling methods of a component and depending on the language; exceptions may be thrown when an error occurs. It is to be noted that the problems cannot be given suitable solutions using contracts and certain exceptions caused by certain problems always cannot be solved by contracts in the fullest perspective. However, a suitable methodology can be employed using contracts.

An illegal call of a method cannot be prohibited by a correct signature. Some pre-conditions often need to be fulfilled by methods before they can be used. After a successful call the result has to match a post-condition and the state of the component may have changed by that time.

Contracts are defined by signatures and a possible sequence of method-calls. Because of the existence of many variants, it may not be possible to describe a correct sequence. A good design should prevent complex sequences and limit the amount of possible variants.

It is necessary to correctly define the start- and the end-state for each method-call in order to describe the correct sequence which is actually a recommended approach. In some cases it is useful to consider the intermediate states. The description can be defined as set of states combined with a set of methods and transitions between them [11].

- S set of states
- $S_s \subset S$ set of start states
- $S_e \subset S$ set of end states
- M set of methods
- $S_s \times M \rightarrow S_e$

In this model transitions correspond to method calls. Classification of contracts into several types can be done by taking a view at the time line of permissible method-calls. This will be modeled by using a state-machine.

4. State-machines as sequence-models

Analyses of the several design processes have shown that state-machines are well-known constructs to control different processes. They are defined by states and transitions between these states. A state-machine does not remember the earlier states but it only knows the actual state i.e., the state which is under consideration [12]

This behavior is not enough for modeling the allowed sequences of method calls and there are other limitations, so that it is necessary to enhance the functionality of state-machines.

4.1 Limitations of state-machines

Before defining a more enhanced state-machine it is imperative to understand their limitations. The limitations of state machines is a possible constraint that needs to be recognized from the research point of view [13]

A one-way state machine exhibiting the one-way-limitation is indicated in the figure 3 below:

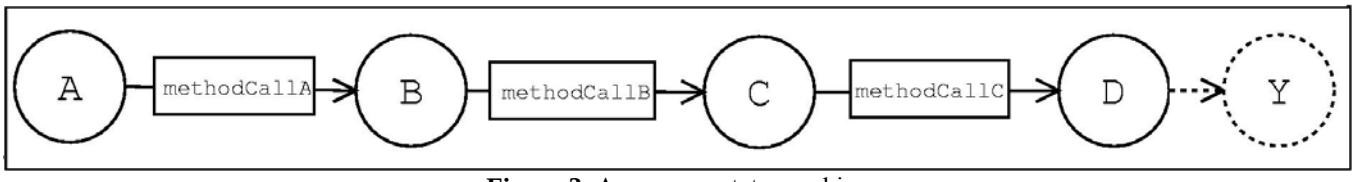

Figure 3: A one-way state-machine

A study on the existing system has shown that a state-machine allows only one way of applying the transitions. Parallel transitions cannot be done in any other way or manner. It would be always better to specify the correct sequence of calls but sometimes it is not possible to specify only one allowed way. Figure 3 shows a sequence of transitions that result in state Y. Reaching state Y in this example is only possible going from state A to state B, etc.

Our research has shown that sometimes a state could be reached when several other states have been reached without defining an order. Our research study has shown that such sequences will quickly get too complex to handle only when State machine permits such an order. In order to avoid this situation, a new view has been incorporated as a result of our research work which resulted in the successful implementation of a one-way state machine which will reach the final state irrespective of the order whether well defined or which needs to be defined in order to reach the final state.

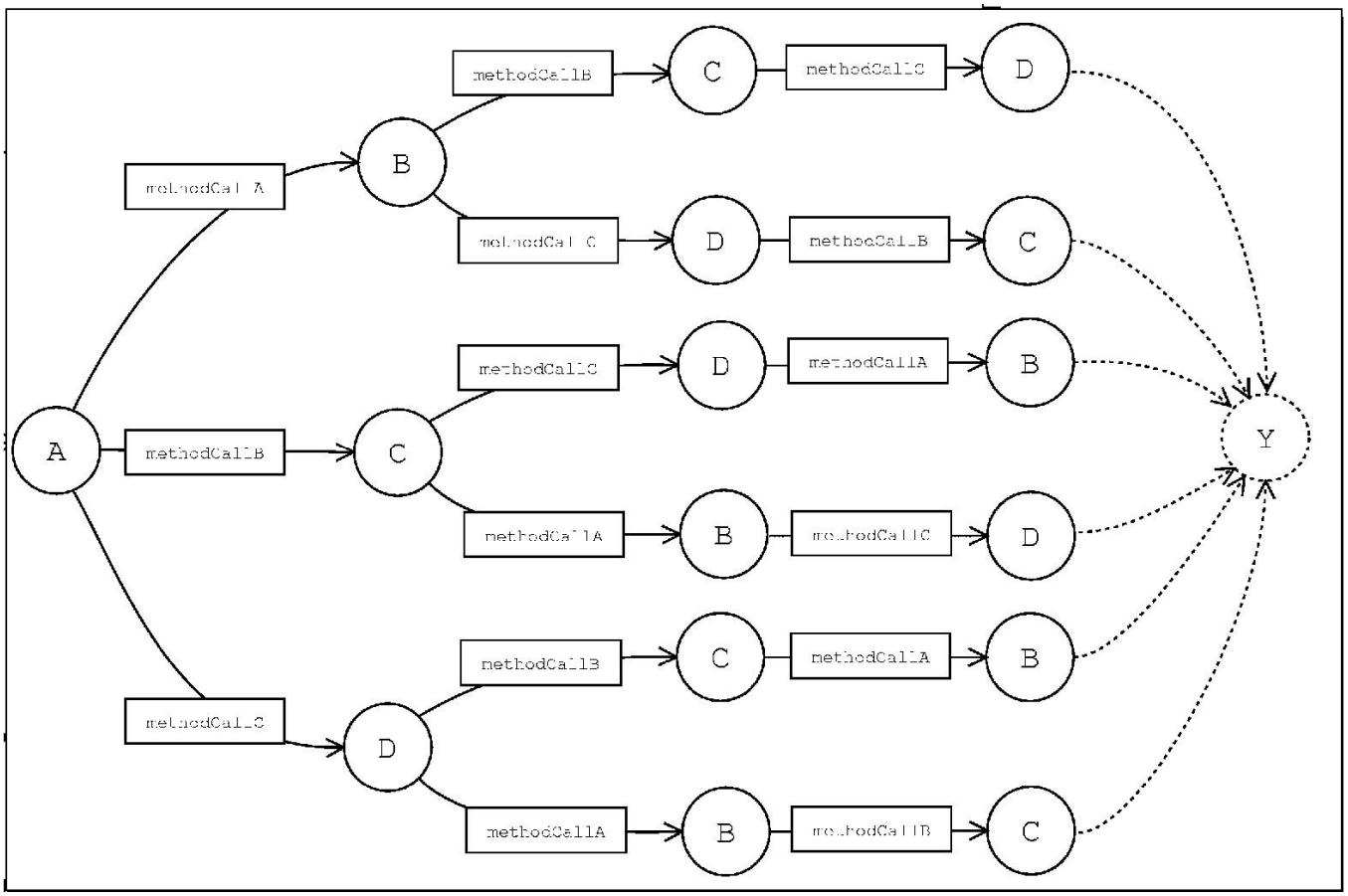

Figure 4: State-machine for permutations of calls

4.2 Defining Groups of States

While considering a group of states as a suitable solution to this problem can be building groups of states. However, state-machines do not support groups of states because of introduction of multiple policies making the system more and more complex. There must be policies for which transitions are valid within the group and which cannot be applied. These policies will get less complex if they can be defined for state-machines as in Figure 4.

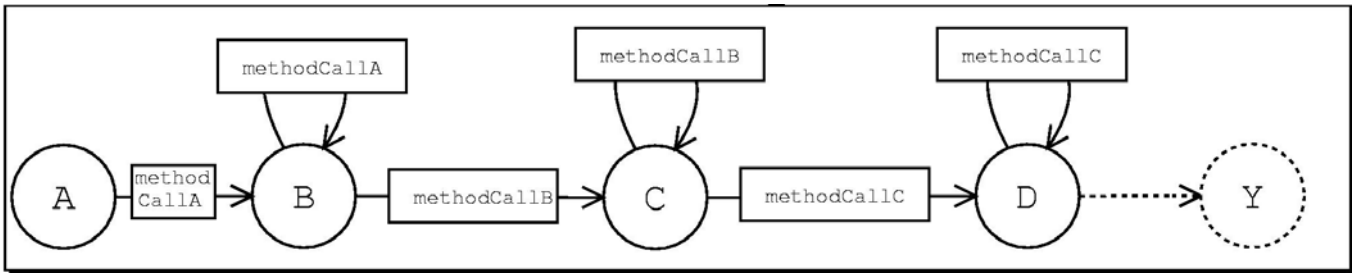

Figure 5: State-machine similar to a one- way state-machine

All transitions that result in a new state can be called in this state again in case of this state machine.

Modeling of components by such state-machines comprise of methods that can be called at least two times without resulting in an error. This might not apply for all methods but a lot of methods show this behavior. In the case of building up of a component, it might be necessary to store references of other components by calling a method. This method can be called several times without causing an error. An example for a method that does not have such a behavior could be the opening of a file containing the setup information. An exception can occur when a method is called twice and when the file is still opened. Our research study has shown that a robust implementation is to avoid such an exception and to make it possible to call the method more than once.

5. Proposed System

5.1 Enhanced state-machines

The proposed system consists of enhanced state-machines that allow to test contracts as defined by using a new set of states. It supports a group of states that are recognized and suitable to implement successfully. All transitions that are allowed between these states can be applied and reassigned to the system many number of times. A method that causes a

switch to a new state but cannot be called again in this state can also be called once in these state-machines. Therefore, this is a new and enhanced way to implement various method calls and contracts successfully by recognizing and grouping states.

After a state-machine has been defined, this enhanced functionality can be applied by grouping states.

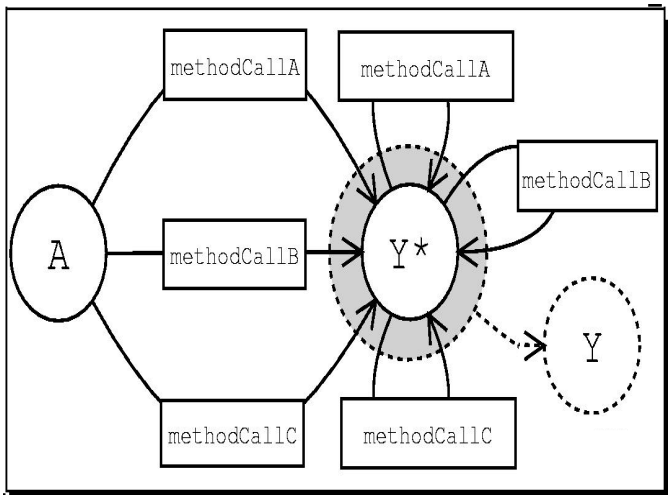

Figure 6: An enhanced state-machine

5.2 The categorization of contracts

This Research paper stresses upon the contracts and their categorization in order to successfully implement them in the proposed system prototype for the purpose of simplicity and minimizing the errors in the system. There are various categories of contracts. For implementing systems that deal with contracts, it is important to define categories for these types. Some types do not need additional algorithms but others need loop-detectors or other algorithms based on graph-theory.

The following are the different types of contracts which need improvisation in the existing system and then incorporate them in the proposed system for the purpose of research study.

5.3 Linear Contracts

Linear contracts are the simplest type. Each state has only one transition that ends in another state.

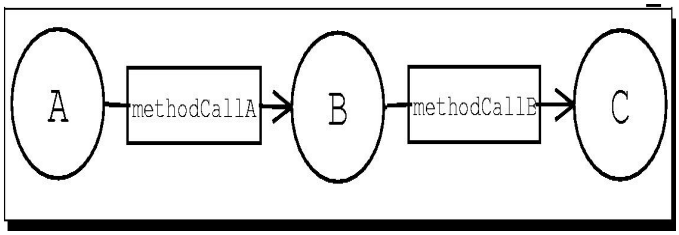

Figure 7: Linear contract

It is very simple to handle such contracts since there is a clear way of how a component that supports this type of contracts has to be used.

5.4 Pseudo Linear Contracts

Pseudo Linear Contracts are somewhat more complex than the linear type of contracts. One or more states have one or more transitions that end either in the same state or in a state that has not been visited.

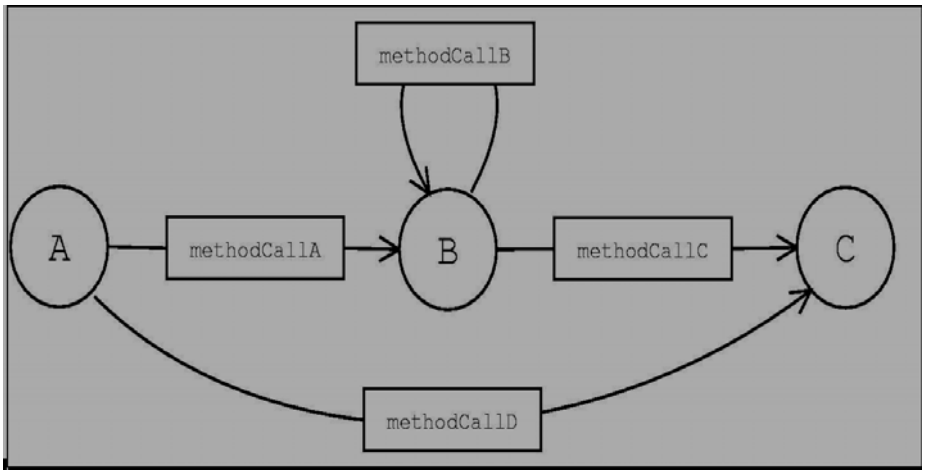

Figure 8: A pseudo linear sequence

A correct perspective has to be established to build a system that will be able to handle such contracts which would be able to solve two major critical issues:

- There are different ways of reaching a state.
- How often a transition is called that ends in the same state.

5.5 Looped Contracts

Our research study has shown that it is necessary for developers to avoid writing contracts that have this behavior as far as possible but when the situation prompts sometimes, it becomes necessary for a component to have such a contract.

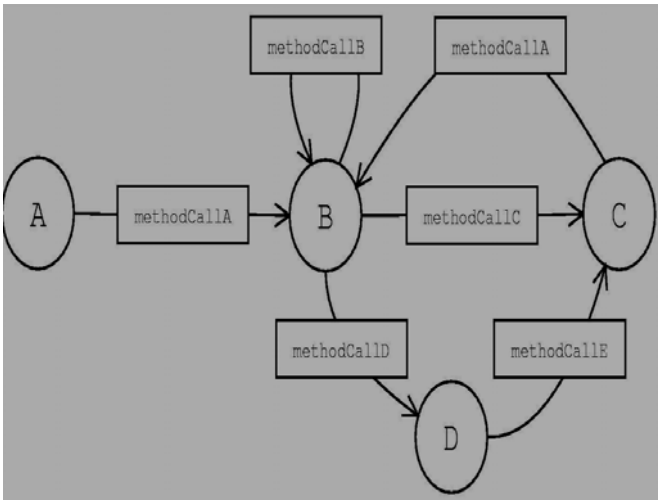

Figure 9: A looped sequence

Our Proposed System is designed for these contracts for the purpose of solving several problems by answering the following questions and possibilities:

- What are the various ways of reaching a state?
- What is the probability that a transition called several times ends in the same state.
- What is the procedure to detect loops?
- How to detect all ways to reach a state.
- How to ascertain the shortest way to reach a state.
- How to identify the various ways of reaching a state.
- How often a transition is called that ends in the same state needs to be known.

6. Analysis and Interpretation

Our proposed model for testing software components uses pre-conditions to define what is to be done before a component can be used. Pre-conditions are often defined in a very formal way resulting in lots of work to define them and involve a lot of analytical work and logical reasoning for interpreting the results accurately.

It is always necessary to define the states during the design of the component itself. The objective is to reduce the complexity of the design component and make it as simple and functional as possible which will in turn result in minimization of errors and create an optimum environment with increased efficiency and functionality.

This research paper lays emphasis on the functionality and operability of the various design components and utilizing the methods calls and contracts from a new viewpoint and also to establish a purposeful and meaningful relationship between states and preconditions. Our research has given the following outcomes as a result of meaningful analysis and interpretation:

- Methods that need the same pre-conditions are in the same state.
- The summary of all states that are needed to reach a state can be interpreted as the pre-conditions of this state.

It is always most appropriate to derive most pre-conditions by utilizing the states. However, this has not been proven and evaluating this assumption and developing an algorithm should be part of further research.

Our research also focuses on the assumption that the manner in which the components will be used in combination with other components. It is to be ascertained that all these components have contracts that must be fulfilled. The components are connected by applying method-calls on each other. The component that applies a method-call on another component is called caller and this is prerequisite requirement for the successful establishment of a connection.

As described in this research paper, there are several types of method calls, deterministic calls and nondeterministic calls. Figures 10 and 11 shows the model of combining components by applying deterministic call as well as for a non-deterministic call.

This is a matter of great simplicity because by invoking a deterministic call will always result in an end-state. Image shows the model of a deterministic call between two components. Both involved components change into their end-states.

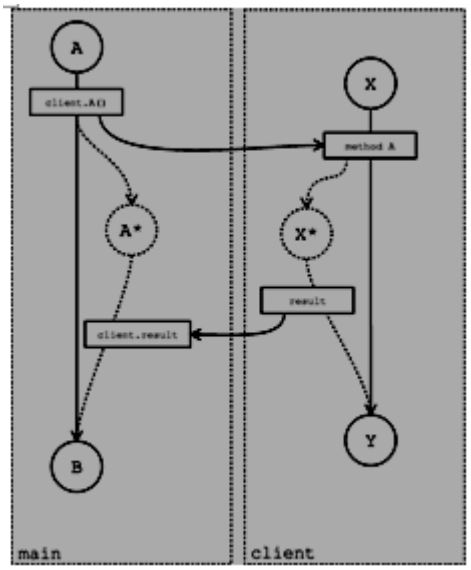

Figure 10: A call of Deterministic Method

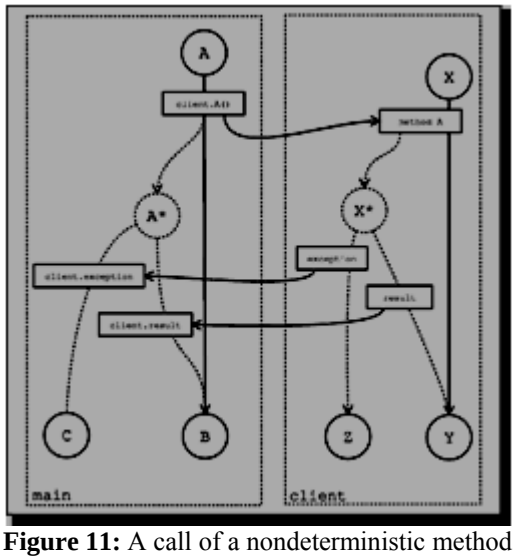

Figure 11: A call of a nondeterministic method

Since there is more than one possible end-state, it becomes more complex in the case of calling a non deterministic method. It becomes imperative the two components must handle all possible exceptions and switch into the corresponding state as shown above.

The following state machines were developed for the research purpose for a better and more methodical understanding which explain both about deterministic and non deterministic method calls and give a broader perspective of the research study. These figures also provide a pathway to software developers to develop suitable prototypes for industrial and pharmaceutical companies which can enhance their software engineering requirements wherein it will lead to advanced productivity and turnover.

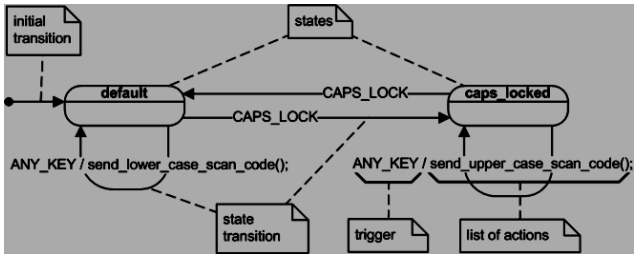

Figure 12: Figure showing interconnected components using method calls

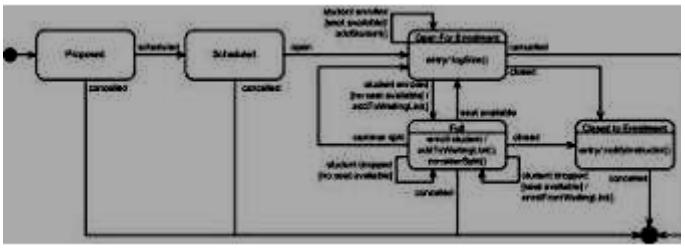

Figure 13: State machine prototype model

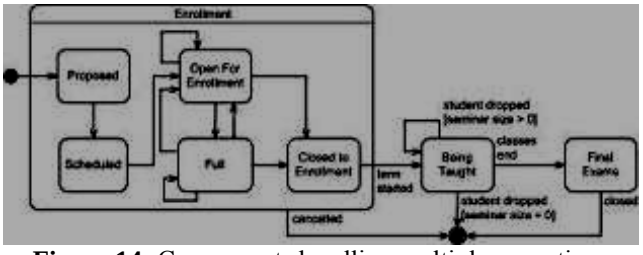

Figure 14: Components handling multiple exceptions

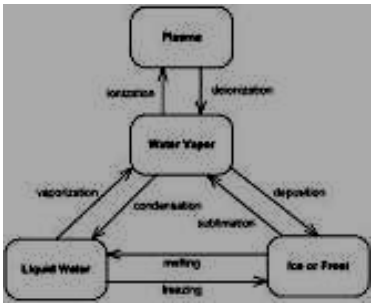

Figure 15: State machine prototype for industrial purpose

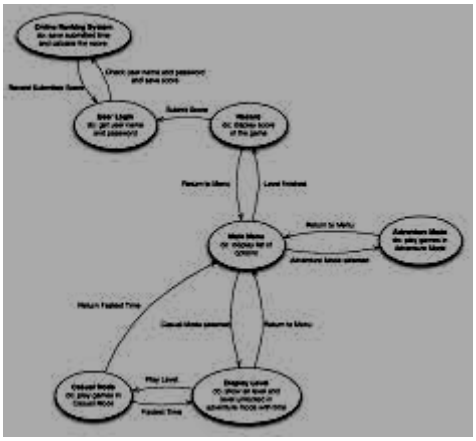

Figure 16: Prototype of a state machine with suitable attributes

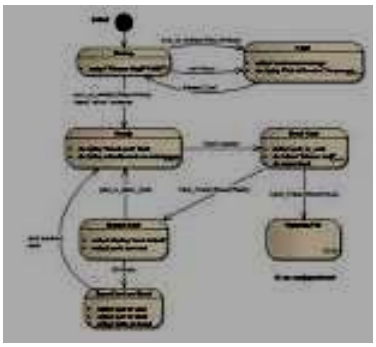

Figure 17: Prototype of a State machine process model

7. Results

The research paper provides a pathway for the software developers in successfully implementing the various method calls and contracts in the assessment of process centered and process driven software engineering environments that rely on process models using suitable method calls and contracts to configure and control their operations in the fullest sense. The results have been excellent in providing several avenues for the improvement and enhancement of these different methods from the existing system and creating a suitable model for the present system. This research study provides several opportunities available for the designers and developers which include:

1) In the first place software developers can think of software process simulation efforts which seek to determine or experimentally evaluate the performance of classic or operational process models using a sample of alternative parameter configurations or empirically derived process data and simulation of empirically derived models of software evolution or evolutionary processes also offer new avenues for exploration as shown in the figures above.
2) Also, the developers can view from the point of Web based applications wherein Web-based software process models and process engineering environments seek to provide software development workspaces and project support capabilities that are tied to adaptive process models.
3) The research lays emphasis on the software process and business process reengineering which focus attention to opportunities that emerge when the tools, techniques, and

concepts for each disciplined are combined to their relative advantage. As such, this will provide in turn giving rise to new techniques for redesigning, situating, and optimizing software process models for specific organizational and system development settings.

4) This research paper provides for understanding, capturing, and operationalizing process models that characterize the practices and patterns of globally distributed software development associated with open source software as well as other emerging software development processes, such as extreme programming and Web-based virtual software development enterprises or workspaces.

8. Conclusion and Future Scope

Considering the fact from the research point of view, it is concluded that code contracts provide a way to specify preconditions, post conditions, and object invariants in the developed code. Also, preconditions are requirements that must be met when entering a method or property. At the same time, post conditions describe expectations at the time the method or property code exits. It is also confirmed that object invariants describe the expected state for a class that is in a good state. Our research has also proved that code contracts include classes for marking your code a static analyzer for compile-time analysis and a runtime analyzer. The classes for code contracts can be found in the System. Diagnostics. Contracts namespace.

The benefits of code contracts include the following:

- Improved testing: Code contracts provide static contract verification, runtime checking, and documentation generation.
- Automatic testing tools: For the purpose of establishing more and more meaningful unit tests, code contracts can play an important role to provide for more meaningful units by filtering out those meaningless test arguments which do not satisfy the required preconditions.
- Static verification: Violations can be checked without running the program with the help of static checker which can decide whether there are any contracts. It checks for implicit contracts such as null dereferences and array bounds and explicit contracts.
- Reference documentation: The contract on formation has to be filed along with the documentation generator augments and existing XML documentation files. There are also style sheets that can be utilized along with Sandcastle so that the generated documentation pages have contract sections.

Technically from the developer point of view, all .NET Framework languages can immediately take advantage of contracts and also there is no need to have to write a special parser or compiler. A Visual Studio add-in lets us specify the level of code contract analysis to be performed. Therefore, the analyzers can confirm that the contracts are well-formed (type checking and name resolution) and can produce a compiled form of the contracts in Microsoft intermediate language (MSIL) format. Authoring contracts in Visual Studio lets us take advantage of the standard IntelliSense provided by the tool which is a plus point.

The research work has also shown that most methods in the contract class are conditionally compiled wherein the compiler emits calls to these methods only when a special symbol is defined namely CONTRACTS FULL, by using the #define directive. CONTRACTS FULL lets us write contracts in the code to be developed without using #ifdef directives. Our research has also shown that it is possible to produce different builds some with contracts and some without contracts.

From the point of view of our research work, all methods that are called within a contract must be pure and as such they must not update any preexisting state. A pure method is allowed to modify objects that have been created after entry into the pure method.

Assuming that the following code elements are pure, the code contract tools normally make use of:

- Methods that are marked with the Pure Attribute.
- Types that are marked with the Pure Attribute (the attribute applies to all the type's methods).
- Property gets accessors.
- Operators (static methods whose names start with "op", and that have one or two parameters and a non-void return type).
- An appropriate method whose properly assigned name starts with "System.String", "System.IO.Path", or "System.Type".
- "System.Diagnostics.Contracts.Contract",
- Any invoked delegate provided that the delegate type itself is attributed with the PureAttribute. The delegate types System.Predicate<T> andSystem.Comparison<T> are considered pure [3]

Our research has indicated that all members mentioned in a contract must be at least as visible as the method in which they appear. For example, a private field cannot be mentioned in a precondition for a public method and clients cannot validate such a contract before they call the method. Anyhow, if the field is marked with the ContractPublicPropertyNameAttribute, then invariably it is exempt from these rules as such.

We can express precondition which a state whenever a method is invoked. By using the Contract.Requires method. They are basically meant to specify valid parameter values. All members that are mentioned in preconditions must be as accessible as the method itself and on the other hand there is every possibility that the precondition might not be understood by all callers of a method. It is also ascertained that the condition must have no side-effects. It is to be noted that the run-time behavior of failed preconditions is determined by the runtime analyzer exclusively.

References

[1] Ambriola, V., R. Conradi and A. Fuggetta, Assessing process-centered software engineering environments, ACM Trans. Softw. Eng. Methodol. 6, 3, 283-328, 1997.

[2] Balzer, R., Transformational Implementation: An Example, IEEE Trans. Software Engineering, 7, 1, 3-14, 1981.
[3] Balzer, R., A 15 Year Perspective on Automatic Programming, IEEE Trans. Software Engineering, 11, 11, 1257-1267, 1985.
[4] Balzer, R., T. Cheatham, and C. Green, Software Technology in the 1990's: Using a New Paradigm, Computer, 16, 11, 39-46, 1983.
[5] Basili, V.R. and H.D. Rombach, The TAME Project: Towards Improvement-Oriented Software
[6] Environments, IEEE Trans. Soft. Engr., 14, 6, 759-773, 1988.
[7] Basili, V. R., and A. J. Turner, Iterative Enhancement: A Practical Technique for Software Development, IEEE Trans. Software Engineering, 1, 4, 390-396, 1975.
[8] Batory, D., V. Singhal, J. Thomas, S. Dasari, B. Geraci, M. Sirkin, The GenVoca model of software-system generators, IEEE Software, 11(5), 89-94, September 1994.
[9] Bauer, F. L., Programming as an Evolutionary Process, Proc. 2nd. Intern. Conf. Software Engineering, IEEE Computer Society, 223-234, January, 1976.
[10] Beck, K. Extreme Programming Explained, Addison-Wesley, Palo Alto, CA, 1999.
[11] Bendifallah, S., and W. Scacchi, Understanding Software Maintenance Work, IEEE Trans.Software Engineering, 13,3, 311-323, 1987.
[12] Bendifallah, S. and W. Scacchi, Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork, Proc. 11th. Intern. Conf. Software Engineering, IEEE Computer Society, 260-270, 1989.
[13] Biggerstaff, T., and A. Perlis (eds.), Special Issues on Software Reusability, IEEE Trans. Software Engineering, 10, 5, 1984.
[14] Boehm, B., Software Engineering, IEEE Trans. Computer, C-25, 12, 1226-1241, 1976.